Httpd

Table of contents

1. CryptoServer

XLattice's CryptoServer is a lightweight Web server for the truly paranoid: nothing is displayed without being authorized and having its integrity guaranteed, and any user interaction with the server can be authenticated and logged.

Note:
The name **CryptoServer** is provisional.

The CryptoServer is a daemon which listens for HTTP requests on a single port, normally port 80. When a request is received, the daemon first checks the source of the request against a blacklist. If there is a match, the connection is silently closed. Otherwise the server checks the name or IP address to which the request is addressed against a list of site names. If there is no match, the connection is dropped. Otherwise the server checks the file requested against a list for that site. If the file name is not present, the server returns a **404** message, "file not found". Otherwise the request is valid and the server attempts to return the file from an in-memory cache. If the file is not actually present in memory, the memory cache retrieves it from a disk cache. If the file is not present there, the disk cache retrieves it from another server, usually using TCP/IP, the standard Internet protocol for reliable communications.

At each step in the process there are cryptographic checks to ensure the integrity of the data being returned.

1.1. Hashes

Except where otherwise qualified, by **hash** we mean a 160-bit binary value. This 20-byte value is most often a **content key,** an SHA-1 digest of a file's contents. SHA-1 is one of the two most commonly used algorithms for generating hashes from binary data; the other, MD5, is believed to be less secure. SHA-1 is quite widely used in business and mandated by the US government for use in communicating and storing sensitive Federal data. Like the title keys discussed below, content keys are guaranteed to be unique: given a document with a particular SHA-1 digest, it is for all practical purposes impossible to find another document with the same digest.

More precisely these hashes are **quasi-unique:** *there is a theoretical possibility that two files or build lists will have the same hash, but this is exceedingly unlikely to occur within the lifespan of the universe.*

The other type of XLattice hash is a **title key.** The integrity of a file with a content key is determined simply by running the SHA-1 algorithm on its contents: if the result doesn't match the SHA-1 digest that is the basis of the file name, the file has been tampered with or is otherwise corrupt. This is good for invariant data but not where you want to assign the same name to data that changes from time to time.

A title key is an SHA-1 digest obtained from the author's public key, normally a 1024 bit quantity, and the document title, where the "title" will most often be a directory name. The title key can identify a single document but more often identifies a particular **build file,** a dated and digitally signed list of names and their hashes.

A file is retrieved from disk or the Net using its hash. The file is identified in storage by that hash. In the simplest implementation, the file name is the hash itself, base-64 encoded. This scheme encodes 6 bits into a byte, so that encoding three bytes of binary data gives you four bytes of printable ASCII text. A 20-byte hash therefore encodes into 28 bytes.

1.2. Cryptographic Identities

In the sense in which we use the term, a cryptographic identity is a pair of keys. One of these, the **public key,** is published and so can be used to identify an entity (a human being, XLattice node, authority, whatever). The other, the **private key,** is held secret. The owner of the key pair can use the private key to encrypt data or digitally sign a document. Using the public key, anyone else can verify that the document was signed or encrypted by the author. Alternatively, anyone can use the public key to encrypt messages which can only be decrypted by the holder of the private key.

The security of systems employing such key pairs is assumed to be related to the length of the keys, with each additional bit doubling the cryptographic strength. Currently keys are usually 1024 bits long.

There are two such systems in common use, DSA and RSA. RSA is generally used both for digital signatures and for message encryption. DSA is normally used only for digital signatures.

1.3. Master Domain, Master Key

When the CryptoServer first boots it is assigned a **master domain** and may be assigned a **master key.** These assignments are made either in a configuration file or on the command line.

The **master domain** is a qualified Internet domain name such as `xlattice.org` or `planetlink.ltd.uk.` The master domain may not be a top level domain (TLD) such as `.com, .net,` or `.org,` nor may it be a so-called CCTLD *(country code TLD)* like `.fr` or `.de.` Nor may it be a domain managed by the national registry like `.co.uk` or `.ltd.uk.` The domain name must be one managed by whoever operates the CryptoServer. It can of course be that of the server itself.

The **master key** is a RSA public key associated with the master domain. It can either be set in the CryptoServer configuration file or retrieved from the Internet domain name system, the global DNS. The public key is normally a 1024-bit value. This key will be used to validate digital signatures on configuration files and control messages sent to the server.

If it is to be obtained from the DNS, the master key must be put in a text record associated with the master domain (see RFC 1035). This will contain a quoted string representing the master key's value, base-64 encoded.

The private key corresponding to the master key must **not** be present on the server. Specifically, the master key is not the same as the public key of the XLattice node on the CryptoServer.

The public key is made visible in the global DNS by adding a line like

```
www.xlattice.org.  IN TXT "rsa 1234567890...abcdefghijklmnopq 3"
```

to the domain's name server configuration file. This is a **Resource Record** or **TXT RR.** The value after *rsa* is a base-64 encoded RSA public key, the master key. If the key is 1024 bits (128 bytes), then that part of the encoded key is 178 bytes long.

By default the master domain is `www.xlattice.org` and by default the master key is the www.xlattice.org public key.

Changes to the DNS can take a long time to propagate over the Internet, weeks in extreme cases. Because of this it might sometimes be advantageous to reserve a set of names just for announcing keys.

```
key1.xlattice.org.  IN TXT "rsa NNNN...NNNN1 3"
key2.xlattice.org.  IN TXT "rsa NNNN...NNNN2 3"
```

1.4. Filters: Blacklists and Whitelists

A **filter** consists of a set of IP address ranges and a list of domain names. If the filter is used to reject matching hosts, it is a **blacklist.** If the filter is used as a basis for deciding that a request should be served, it is a **whitelist.**

Address ranges are inclusive: A source address is in range if it is equal to the low-order IP address in the range, the high-order address, or anything in between. If the filter is a blacklist and the IP address from which a request comes is in one of the blacklisted ranges, the connection is silently closed.

Domain names are matched from right to left. In making the comparison, names are divided into parts at the dots and only the parts are compared. If a source address contains or equals a blacklisted domain, the connection is dropped. So if for example `xyz.com` were to be blacklisted, then `xyz.com` and `abc.xyz.com` would be rejected, but `xyz1.com` and `wxyz.com` would not be.

In the longer run, the intention is to automatically detect abusive contacts and add them to the blacklist. However this functionality will not be implemented soon.

The blacklist is loaded when the daemon boots. The blacklist is digitally signed using the SHA1withRSA algorithm, which uses the CryptoServer's master key.

1.5. Site List

A **site list** is a build file (see below) containing a list of domain names. The site list is signed using the private key corresponding to the CryptoServer's master key.

The site list is a list of the Web sites supported by the CryptoServer.

The title of the site list is **site.cfg.** Given this title and the RSA public key, it is straightforward to determine the title key of the site list and retrieve it, if it is not otherwise available.

1.6. Build Files

A **build file** consists of

- a title line
- a line holding the base-64 encoded public key of the signer
- a timestamp
- zero or more data lines containing the extended hash of a file followed by the path to the file relative to the title
- a blank line
- and a digital signature

Each line is CRLF-terminated. That is, each line ends with an ASCII-13 character (carriage return) followed by ASCII-10 (line feed).

The title line is commonly a UNIX-style path using a single slash ('/') as the separator. The path is **not** terminated with a separator.

If the build file represents a subdirectory tree, then the path is prepended to the file names in the data lines. Each such name is a path relative to the title and must begin with a UNIX-style separator, a forward slash ('/').

If the build file represents a site list, the title line is the path to the directory containing the build files for each site.

The digital signature is generated from the concatenated title, public key, timestamp, and data lines, if any, including their terminating CRLFs.

The build file itself has a hash, the **title key.** This is the SHA-1 digest of a function of the title and the signer's public key. Because of the way in which the hash is calculated (it is independent of the timestamp, content, and digital signature line), this is a permanent, globally unique identifier for the build list. When XLattice key-based storage systems are presented with a candidate replacement for a build list, they should only replace the existing build file under its title key if

- the replacement has the same owner (public key line) and title

- the replacement has a valid digital signature
- and the replacement has a more recent time stamp

Current build lists are stored twice, once by title key and once by content key. When the build list is replaced by a newer list, the version stored by title key is replaced, but not the version stored by content key.

1.7. In-memory Cache

The CryptoServer should normally serve Web pages from its in-memory cache. The cache should be large enough so that most (say 90% or more) requested pages are in it. The cache is read-through, meaning that if a page is not found it is read into the cache before being served. Pages are always served from the cache.

If the cache fills up, infrequently used pages are dropped from the cache using an **LRU** (least recently used) algorithm.

1.8. Disk Cache

Pages not found in the in-memory cache are fetched from the disk cache manager. Any page retrieved is always subject to an integrity check. If it has a content hash, the disk manager first reads the file into memory and then takes an SHA-1 hash of its contents, verifying the SHA-1 hash against the content hash. If the file is a build list,

- its hash (a function of public key and title) is verified
- and then its digital signature is verified

In either case, if verification succeeds, the file is passed to the in-memory cache manager. Otherwise the file is discarded and the disk file deleted.

In such cases and also where the file is simply not present in the disk cache, the data is retrieved from the network using the hash as a key. The disk cache is also read-through and managed using an LRU algorithm: data retrieved from the Net will be written to disk before being passed to the memory cache manager, and if there is insufficient room, the least recently used data on the disk will be removed to make room for the new data.

1.9. Retrieval from the Net

Where a file is not found in the disk cache, it is fetched from the Net using its hash as a key. The Web site's owner, the signer of the build list, will normally be an XLattice Peer of the CryptoServer.

It is likely that for most early applications the entire Web site will be loaded directly into the CryptoServer's disk cache or will be fetched from a single backing server. However, as XLattice's routing protocols are developed, it will become possible for the server to fetch files by key from arbitrarily complex networks.

1.10. HTTP Protocol Version Support

CryptoServer will first support HTTP/0.9, then HTTP/1.0 as defined by RFC 1945 , and then HTTP/1.1 as specified in RFC 2616 . More detailed information on the implementation plan can be found in the Roadmap .

1.11. Dependencies

The server software depends upon a number of XLattice components and other jars. At the time of writing these are

- util-0.3. **x** .jar
- xpp3-1.1.3.4.C.jar
- corexml-0.3.x.jar
- crypto-0.0. **x** .jar
- node-0.0. **x** .jar
- protocol-0.0. **x** .jar
- overlay-0.0. **x** .jar

The 'x' in the XLattice component version number is currently 1 in each case, but this will be changing frequently as implementation progresses. The xpp3 jar is an XML pull parser.

Current jars for all dependencies will be included in the CryptoServer distribution, excepting only Java's runtime libraries.

1.12. Security

Because the private keys used to sign build files should not be present on the CryptoServer, it should not be possible to modify the server's configuration or any of the files being served without this being detected. When the server detects that a file has been modified, it will attempt to replace it from a secure source. If it cannot, it will either do nothing (fail to respond to the request) or shut down entirely.

This means that the server should be secure against all but very determined attackers with very large resources. If the attacker has complete control of the server and the necessary technical skills, he or she could simply replace the server or its software entirely. However, it is very likely that this would be quickly detected.

More subtle changes would require compromising the keys used to sign configuration files. This is for all practical purposes impossible so long as the operator of the CryptoServer takes realistic steps to safeguard the physical security of these keys.